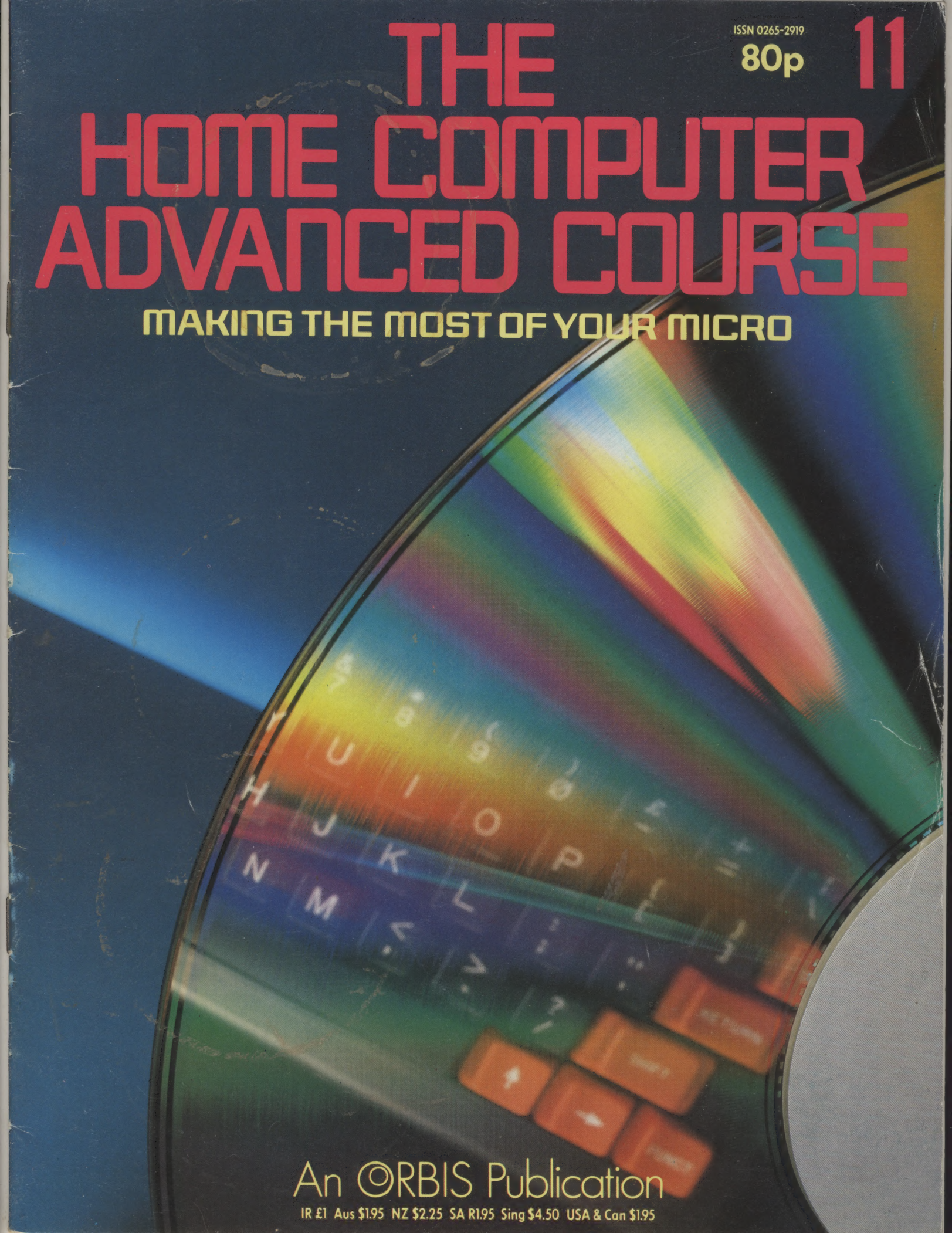
ISSN 0265-2919

80p 11

# THE HOME COMPUTER ADVANCED COURSE

## MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

# CONTENTS


APPLICATION

**LIGHT REVOLVER** Laser discs are now within the budget of the micro owner 201

HARDWARE

**MAKING MOVES** The Sharp PC-5000 is an innovatory portable micro 210

SOFTWARE

**ORDERLY PROCESSION** We learn how to create our own sequential files 204

**FINAL REPORT** Concluding our series on business software for home micros 212

COMPUTER SCIENCE

**YOUR NUMBER'S UP** We design a binary to digital display converter 206

JARGON

**FROM BREAK TO BUS** A weekly glossary of computing terms 208

PROGRAMMING PROJECTS

**GO SUB GO!** We begin a new series on graphics for all the popular micros 214

MACHINE CODE

**COUNTER INSTRUCTIONS** Learning how to implement labels and loops is fundamental to efficient programming 216

PROFILE

**BREAKING AWAY** Zilog is the company responsible for the Z80 CPU 220

WORKSHOP

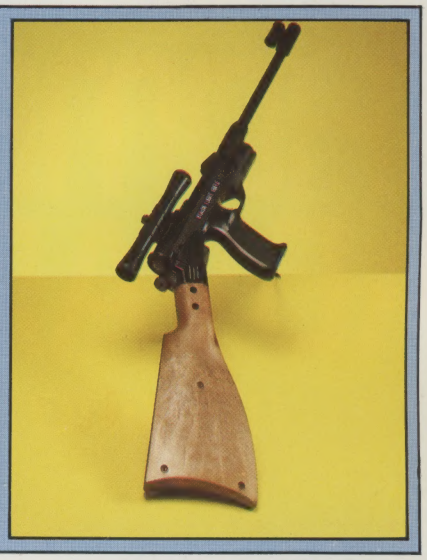
**WORKOUT** Our suggested answers to the problems set on page 194 209


## Next Week

**● The Stack Light Rifle provides a shot of excitement for Spectrum and Commodore users. We take a close look at this unusual peripheral.**

**● Any micro can be linked to other electronic equipment to give you total control. We explore the possibilities of this exciting application.**

**● Digital Research owes its success to the CP/M operating system. We look at how this top software company plans to stay ahead of the competition.**



## QUIZ

**In this section we will be featuring a regular quiz based on the articles in the current issue.**

1) Using 6502 Assembly language, add two numbers stored at memory locations ADR1 and ADR2 and store the result in location ADR3.

2) Perform exactly the same calculation, but this time in Z80 Assembly language.

3) How would you change the screen colour on a Commodore 64 to white with a black border?

4) How many different sprites are available on the Commodore 64

5) What are the three main buses connected to a typical microprocessor?

6) What do the letters CP/M stand for?

**Answers in next week's issue.**

QUIZ

COVER PHOTOGRAPHY BY IAN McKINNELL


**Editor** Max Phillips; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Contributors** Lisa Kelly, Steven Colwill, Geoff Bains, Tony Harrington, Richard Pawson, Mike Wesley; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd., Leicester**

**HOME COMPUTER ADVANCED COURSE** – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

**How to obtain your copies of HOME COMPUTER ADVANCED COURSE** – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

**Back Numbers UK and Eire** – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

**How to obtain binders for HOME COMPUTER ADVANCED COURSE** – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

**Note** – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

# LIGHT REVOLVER

LASERVISION COURTESY OF TELETAPE VIDEO LTD

IAN McKINNELL

**Floppy disks and video discs are high-density direct-access storage devices that until recently have been too expensive for the home enthusiast or small business user. However, increased production and competition in both industries have resulted in a steady fall in price. We review their range of applications for home micros.**

Many people will be surprised to learn that laser disc players, regarded as high-cost luxury items a few years ago, now cost less than video cassette recorders, yet provide far better picture reproduction. A television picture is a dynamic image that can be recorded on a video tape as an unbroken sequence. The only way to find a particular part of the sequence is to wind the tape forward, either at 'play' speed or by using 'fast forward' with the tape counter to guide you. Discs store the images as separate frames and enable you to go directly, accurately and quickly to any one of them. A frame's location on disc can be described in terms of track and sector, and a microprocessor can keep an up-to-date catalogue of the locations. The microprocessor oversees the disc access and can provide freeze-frame or slow motion facilities, and stereophonic sound.

The microprocessor's tasks of driving a turntable at constant high speeds and positioning a playback head precisely at certain positions on the surface are not the major technical achievements of disc technology. The greater challenge was in facilitating the enormously high storage density of the discs. To enable thousands of frames to be stored on a disc the size of a 12″ long playing record was especially daunting.

If you've ever used high resolution graphics on your micro, then you know that a television picture is made up of individual dots (pixels) of light — the more dots per screen the better the picture displayed — and that storing high resolution screen displays uses up a lot of memory. The BBC Micro, for example, has a maximum resolution of

**Laser Lounge**
Laser disc players will soon be as inexpensive as home computers and it won't be unusual to find the two linked together in the home. By buying appropriate discs and software, you will have access to vast pictorial databases and sophisticated training programs, as well as animated adventure games

IAN McKINNELL

**New Memory**
Twelve-inch laser discs and the smaller compact discs can be used for storing both video and audio information as well as the digital data stored on computer cassettes and floppy disks. The drawback is that the discs cannot be written to and are technically ROM only. This means that you cannot save your own programs and information on laser disc. Instead, you will have to buy or hire pre-recorded software

640×256 pixels per screen display, which occupies exactly 20 Kbytes of memory. If a television frame were no finer in resolution than this, then storing one second's worth of video (at 25 frames per second) would require 25×20 Kbytes of storage! A minute of a television programme would occupy 30 Megabytes (30,000,000 bytes), and an episode of your favourite television serial could take over a Gigabyte (1,000 Megabytes) of memory — and on the single-sided single-density floppy disks that most home computers use would require over 6,000 disks and a week's work!

Given these figures, video tape storage becomes rather more attractive: putting half an hour's worth of television programs on a disk seems to pose insurmountable problems.

The answer to this dilemma lies in writing the data very small. A laser disc recorder etches the data with a hair-fine laser beam onto a metallic platter coated in a tough translucent shell. A low-power beam is used to read the data off the disk. Lasers are used as the read-write stylus for this disk because they are such fine-resolution low-tolerance devices. No other technique could read and write so much data in so small a space.

The format is a combination of the techniques used on hi-fis and disk drives. The grooves of a gramophone record form a continuous spiral, and the walls of the grooves carry an etched representation of the recorded sound's waveforms. Most micro users know that the tracks on a disk drive are concentric circles, and that information is written magnetically onto the disk, and stored digitally as patterns of zeros and ones. The laser disc format uses tracks, not grooves, but these form a spiral. Information is etched optically onto the disc in patterns of ones and zeros, but cannot be erased. The ones and zeros (which represent the patterns of dots comprising the television picture) are written on the disc surface by the laser's burning tiny pits into the metal film to represent ones, and leaving it intact to represent zeros. A pit is half a micrometre (0.0005 mm) wide, and one tenth of a micrometre (0.0001 mm) deep. Therefore, a square centimetre of disc surface could hold 400,000,000 of these pits.

This astonishing miniaturisation is just sufficient to cope with the storage demands of video. A 35 cm diameter disc holds 54,000 television frames per side, or roughly 36 minutes of playing time. The calculations of frame size that we made earlier in this article were based on computer graphics resolution in black and white, whereas the laser disc has to store colour information for each dot of the television picture, and carry an audio track as well. A colour frame, with its corresponding sound track, might therefore require 100 Kbytes of storage space. 54,000 such frames would use up 5,400,000 Kbytes — 5.4 Gbytes.

Having overcome the problem of storage limitation, laser discs offer extensive applications possibilities. One major benefit is to remove one of the biggest obstacles in data processing — data collection and entry. Information is usually readily available, but somebody still has to sit down at a terminal and actually type coded representations of the data into the system — a tedious, expensive and time-consuming procedure. If instead you can point a camera at the data, and let that store the visual information on disc while you enter only indexing details of the recorded pictures, then the workload is greatly diminished.

Laser disc players vary greatly in their sophistication. Already available are cheap players for home use. These can be used to show films just like an ordinary video cassette recorder but with the added advantage of tremendous picture quality both in normal use, and in freeze-frames and slow motion playing. You can also select individual frames on some machines by keying their number on a handset and using a search facility. The player will blank the screen for a moment and return with the relevant picture displayed.

The real possibilities don't begin until you can have individual frames selected by a computer program. Players that can do this are available from companies such as Pioneer and Philips, but as yet are expensive and intended only for professional use. The simplest system is to use an IEEE or RS232 interface so that the computer can select a particular frame by its number.

Appropriate software on the computer can keep a list of the available frames on a database and select from them as it needs to.

Philips has taken the idea a step forward and built a simple microcomputer into its more recent models. This can load a program appropriate for a particular disc from either an EPROM cartridge plugged into the machine, or alternatively from off the laser disc itself. Each laser disc stores two audio tracks and one video track. This allows a single disc to have a soundtrack in two languages for example. However, if the second audio track is not needed it can be used to store a computer program.

So we have a bank of 54,000 quality images under the control of the computer. The final stage is to mix the pictures from the laser disc with text from the computer. This could be done with two separate monitors, or by mixing two video inputs, or by using a monitor with its own teletext generator. This final step is an entirely new medium — interactive video. The user and software can guide the television display, both action sequences and still frames, by reading the disc in any order.

The most obvious application is for a pictorial database. The user could ask questions of the computer and it would retrieve relevant information from a database and instruct the player to display an appropriate video frame. This could be used for reference in libraries and schools, for everything from identifying various flowers to selecting items from a catalogue.

Interactive video can progress a stage further by involving the user in the sequences shown on the screen. A training program could explain something with a short clip of film and then ask questions, recapping if necessary or going into more detail and so on. You could even produce films that the user directs by taking decisions on behalf of the characters in the story. The whole course and ending of the film would be different depending on how you 'play' it. Fans of adventure games on home micros should find these new possibilities most absorbing.

There are, of course, problems with developing this market. Apart from the cost of equipment and manufacturing laser discs, new skills have to be developed in designing and producing interactive discs, both in terms of computer software and the scripting and filming of images. Many small companies are beginning to face these challenges and one multi-national, a company called Computer Assisted Televideo (CAT), has already established itself as a dominant force in the business. The company provides a complete service for customers — choosing and installing equipment, designing and producing discs and writing the relevant software. CAT is willing to tackle almost any application of interactive video but high costs currently limit most of its work to producing training programs for large businesses.

However, the possibilities for the home micro owner should not be ignored. Although the cost of producing discs is high, it is no higher than the costs associated with films and recorders. Laser discs already sell for around £10 each so interactive discs could be sold, with software, for around £15 to £20. Considering the quality of entertainment and educational programs made possible by interactive video, this is a very reasonable price indeed.

**Interactive Video**
Many laser disc players have an IEEE or RS232 interface that allows them to be controlled by microcomputers. A typical set-up might be a computer with a database relating to pictures on the laser disc stored on floppy disks. The user can select items of interest from the database, the computer will then instruct the laser disc player to fetch and display the appropriate picture by reference to a frame number. The system shown here adopts a popular approach of combining the computer's output and the laser disc pictures on the one screen. A simpler system would use two separate screens

STEVE CROSS

# ORDERLY PROCESSION

**The sequential (or serial) file is a heritage of the days of tape-based data processing; if magnetic disks had been the cheapest form of mass storage for the last 40 years, then our methods of handling data would be very different indeed. In this article we discover how to create and access sequential files from a BASIC program.**

The binary (or program) file is a simplified case of a sequential file. When you type SAVE"progname", the operating system performs the several discrete operations required in creating a file: first, it opens communication with the tape or disk drive, and writes a header label containing the file name and some information about the file contents. Next, the operating system writes the block of memory containing the current program into the file. Lastly, it writes an end-of-file marker to the file, and closes communication between the computer and the tape or disk drive.

The BASIC commands used in creating a sequential file vary from computer to computer, but they must all carry out the same tasks. Taking the Commodore 64 as an example:

```
100 R$="This is one record of the file"
150 OPEN 5,1,2, "DATAFILE,SEQ,WRITE"
200 PRINT#5,R$
250 CLOSE 5
```

This starts with the OPEN 5,1,2 command, which prompts the operating system to establish a channel of communication, called channel 5, to device number 1 — the tape drive. The operating system can create sequential files on a number of different devices, and it can access several different files at one time, so the channel from computer to device and thence to a particular file must be labelled. (The number 2 is specific to the Commodore and unimportant for the moment.)

After the OPEN command comes the file name. On the Commodore the file name consists of a name such as 'DATAFILE', followed by the file type (SEQ, standing for sequential file) and the access method (WRITE indicating that the file is being OPENed for writing). Sequential files can be OPENed for either writing to or reading from, not both at the same time. The effect of this command is that the device read/write head is energised, and a 'header' record consisting of the file name and some system information is written to mark the start of the file.

The PRINT#5 command in line 200 sends data to the file. PRINT has its usual meaning, but the '#' sign indicates that data is to be sent to the specified channel rather than to the screen — the default output channel. The contents of R$ are, therefore, sent down channel 5 to the sequential file called 'DATAFILE'. The file now contains one record — the string, "This is one record of the file". Finally, CLOSE 5 closes channel 5 to further communication, and writes an end-of-file marker onto the file.

Information in a file is intended to be read at a later stage, for example:

```
500 OPEN 3,1,2, "DATAFILE,SEQ,READ"
550 INPUT#3, A$
600 CLOSE 3
650 PRINT A$
```

Both this fragment of program and the previous one use the OPEN command, here meaning 'find the beginning of the file called DATAFILE and prepare to read from it', whereas previously it meant 'create a file called DATAFILE, and prepare to write to it'. The channel numbers are different, but they are simply labels (a different number could have been chosen in both cases without affecting the operation); the device address, however, is the same in both cases because the file is stored on device 1 — the tape drive. The file name and the file type are the same because they identify the file to be accessed, but the access method is different — READ instead of WRITE.

This program has INPUT#3,A$, meaning 'input from channel 3' instead of from the keyboard — the default input channel. The first complete record in the file is read and sent along channel 3 into the variable A$, just as it was sent along channel 5 from the variable R$ by PRINT#5,R$.

Some of the limitations and most of the advantages of sequential files are thus revealed: they can store anything that a variable can hold, and there are no restrictions on file size or structure. On the other hand, the contents of a sequential file must be read from the start of the file, record by record; there is no way to open the file at a particular record, nor to skip, re-read, delete, insert or append a record.

BOB HALL

**Order Of Play**

Sequential files were developed with tape storage in mind; but the order in which records are sent to the file is the order in which they must remain. The only way to sort or edit a sequential file is by creating a new version of it elsewhere on tape, just as a music cassette can be edited only be re-recording or by physically cutting the tape

# File And Find

```
199 REM++++++++++CBM C64++++++++++
200 REM+       WRITE      FILES      +
201 REM++++++++++CBM C64++++++++++
220 GOSUB 1500
240 FOR K=65 TO 90
260 Z$=CHR$(K)+X$:C$=D$+"WRITING "+CHR$(K)
280 GOSUB 2000
300 NEXT K
399 REM++++++++++++++++++++++++++++
400 REM+        READ     FILES       +
401 REM++++++++++++++++++++++++++++
420 FOR L=0 TO 1 STEP 0:FOR M=1 TO 1
440 PRINT D$;"ACCESS TO RECORDS"
460 INPUT"SEARCH STRING ( *=QUIT )";N$
480 L$=LEFT$(N$,1):IF L$="*" THEN GOTO 5000
500 IF L$<"A" OR L$>"Z" THEN M=0
520 NEXT M
540 Z$=L$+Y$
560 GOSUB 3000
580 PRINT TAB(5)"RECORD ";N;"(HIT ANY KEY)"
600 GET GT$:IF GT$="" THEN 600
620 NEXT L
999 END
1499 REM*****************************
1500 REM*****INITIALISE  S/R********
1501 REM*****************************
1520 D$=CHR$(147):PRINT D$,CHR$(8);CHR$(142)
1540 X$=",S,W":Y$=",S,R"
1600 RETURN
1999 REM*****************************
2000 REM*****WRITE A FILE S/R********
2001 REM*****************************
2020 PRINT C$:INPUT"HOW MANY RECORDS";R
2040 OPEN 8,8,2,Z$
2060 IF R=0 THEN PRINT#8,"*":CLOSE8:RETURN
2080 FOR I=1 TO R
2100 PRINT C$:PRINT "RECORD #";I
2120 INPUT "TEXT.....";R$
2140 PRINT#8,R$
2160 NEXT I
2180 PRINT#8,"*":CLOSE8
2499 RETURN
2999 REM*****************************
3000 REM*****READ  A FILE S/R********
3001 REM*****************************
3020 PRINT D$;"SEARCHING ";L$;" FOR ";N$
3040 OPEN 8,8,2,Z$
3060 FOR I=1 TO 100000
3080 INPUT#8,R$
3100 PRINT R$
3120 IF R$="*" THEN N=0:I=100000
3140 IF R$=N$ THEN N=I:I=100000
3160 NEXT I:CLOSE8:N$=""
3180 RETURN
5000 REM*****CLOSE     PROGRAM********
5020 PRINT CHR$(9);"END OF PROGRAM":STOP
```

**220:** Initialise

**260-280:** Create files "A" to "Z"

**420-540:** Input search record, find initial letter, identify file containing it

**560:**Search that file

**580-600:** Report outcome of search

**1520:** Clear screen, set upper case mode

**2040:** OPEN file to WRITE

**2060:** Check for empty file, write "*", CLOSE file

**2120-2140:** Input text of record, write it to file

**2180:** Write "*" as last record, CLOSE file

**3040:** OPEN file to READ

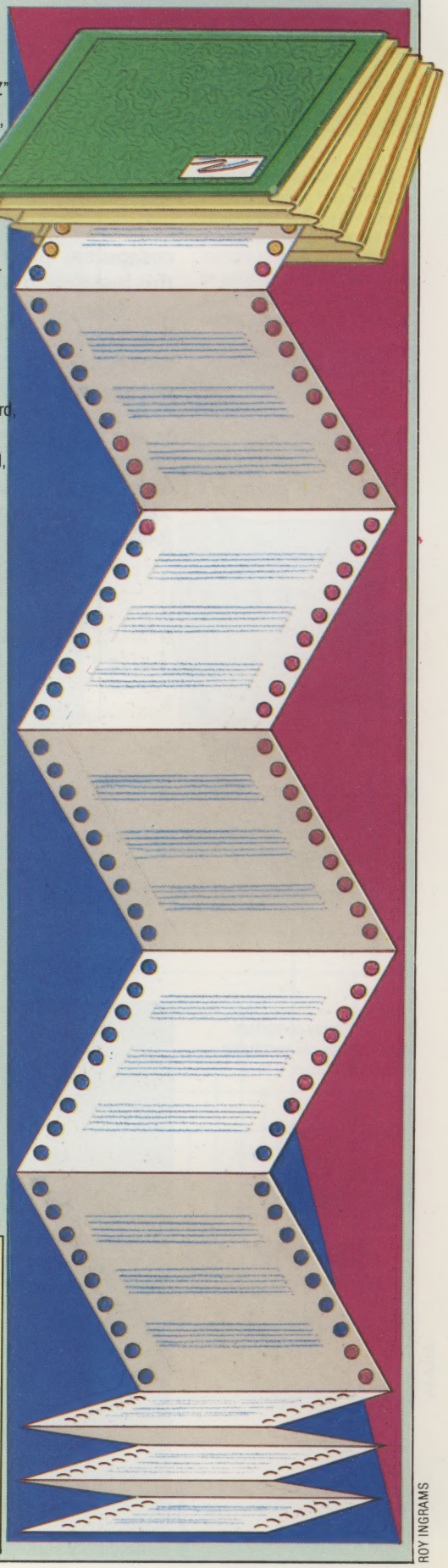
**3120:** Test for last record, quit search

**3140:** Test whether record found, quit search

**3160:** CLOSE file

This program demonstrates the use of sequential files on disk by creating a simple alphabetical index book comprising 26 files, one for each letter of the alphabet. Into each file you can type records beginning with that letter, or no records at all. You can then search for any record. The appropriate file will be searched and displayed until the record is found; if it is not found, the message 'RECORD 0' will be displayed.

The program uses the disk operating system to gain direct access to the file appropriate to the search record; the file itself is then read sequentially, however. This cuts search time to a reasonable length; if the files were on tape, searching would be unbearably slow



ROY INGRAMS

## Basic Flavours

**BBC Micro**

Follow the Spectrum variations for lines 260 and 540
Replace PRINT#8, INPUT#8, and CLOSE8 by PRINT#C8, INPUT#C8, and CLOSE#C8
600 GT$=GET$
1520 *DISK
1530 MODE 7
1540 D$=CHR$(12):PRINT D$"USE UPPER CASE"
1550 PRINT"--HIT ANY KEY--":GT$=GET$
2040 C8=OPENOUT(Z$)
3040 C8=OPENIN(Z$)
5020 PRINT"END OF PROGRAM"

**Spectrum Microdrive**

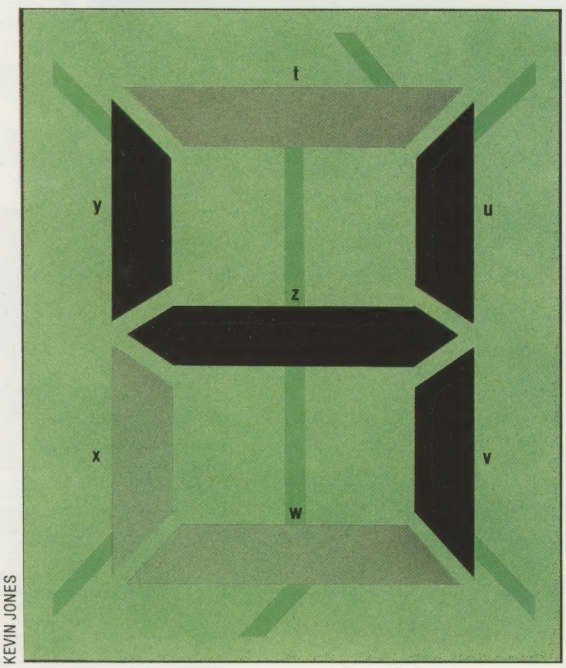
Insert LET where necessary.
Replace PRINT D$;.. by CLS PRINT..
Replace PRINT C$ by CLS:PRINT: C$
Replace OPEN 8,8,2,Z$ by OPEN #8;"m";1;Z$
Replace CLOSE8 by CLOSE #8
Delete line 1540
260 Z$=CHR$(K):LET C$=""WRITING"+Z$
480 LET L$=N$(1):IF L$="*" THEN GOTO 5000
540 LET Z$=L$
600 PAUSE 0
1520 CLS:LET F2=PEEK 23658:POKE 23658,8
3080 INPUT #8;R$
5020 POKE 23658,F2:PRINT "END OF PROGRAM":STOP

# YOUR NUMBER'S UP

**Seven-segment displays are used on calculators, watches and some portable computers to represent decimal digits. In this instalment of the logic course, we design a 'binary to seven-segment display converter' — a circuit that will convert the binary signals used by computers into the decimal digits that humans normally use.**

In our circuit design we shall assume that the binary representations of decimal numbers are in a code known as binary coded decimal, or BCD. That is, each decimal digit is stored as a group of four binary digits, as shown in the table.

| Binary Coded Decimal | Decimal |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 to 1111 | unused |




The LCD display comprises seven individually switchable bars, here labelled 't' to 'z'

KEVIN JONES

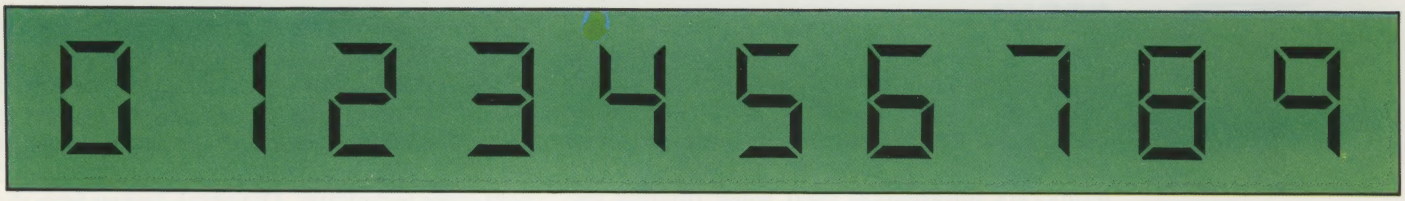
A seven–segment display represents decimal numbers by illuminating certain light-emitting diodes, in the case of an LED display, or by changing the polarity of certain bars (LCDs) in a liquid crystal display. The illustration shows how we have chosen to name the seven segments (the top bar is termed 't', the lower bar 'w', and so on). We also give the various combinations of bars required to form the digits 0 to 9.

This is all the information that we require to assemble a truth table for our converter. We can tell from the illustration on the opposite page what bars need to be activated when the circuit receives each individual input. For example, if the binary input is 0100 (decimal 4), the circuit will activate the bars labelled u, v, y and z. Similarly, the input 1000 (decimal 8) will cause all of the bars to be illuminated. The truth table for the circuit will be:

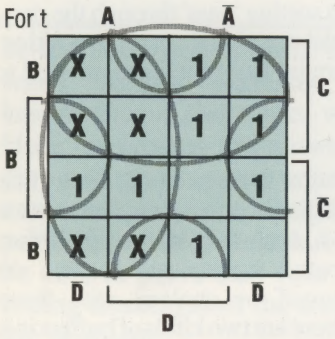
| Decimal | Inputs | | | | Outputs | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | t | u | v | w | x | y | z |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| | 1 | 0 | 1 | 0 | X | X | X | X | X | X | X |
| | 1 | 0 | 1 | 1 | X | X | X | X | X | X | X |
| | 1 | 1 | 0 | 0 | X | X | X | X | X | X | X |
| | 1 | 1 | 0 | 1 | X | X | X | X | X | X | X |
| | 1 | 1 | 1 | 0 | X | X | X | X | X | X | X |
| | 1 | 1 | 1 | 1 | X | X | X | X | X | X | X |

Unfortunately there is no easy way to analyse this truth table, and consequently each output — t, u, v, w, x, y and z — must be simplified separately. We can construct a Karnaugh map for each output, placing on each map the 'don't care' (X) conditions. These may aid simplification in some cases. The seven k-maps, one for each line, are given here. By circling groups, we can produce a simplified expression for each output line. Further simplification is possible in some cases by factorisation.

The first k-map, for example, deals with those inputs that will cause the bar 't' to be activated (this bar is used in all but two of the numbers). Having simplified expressions for all the output lines, we can then draw up the final circuit.

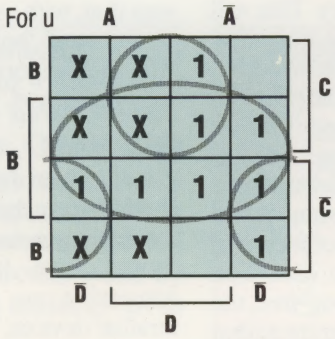



For t



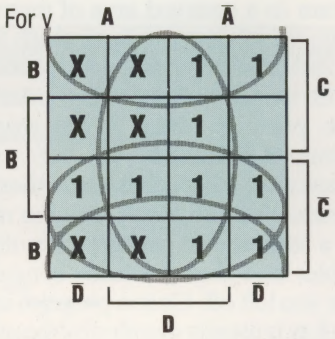

t=C+A+B̄.D̄+B.D

For u




u=B̄+C.D+C̄.D̄

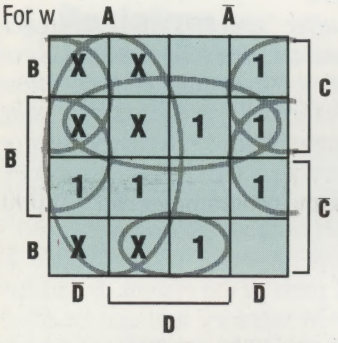
For v




v=D+C̄+B

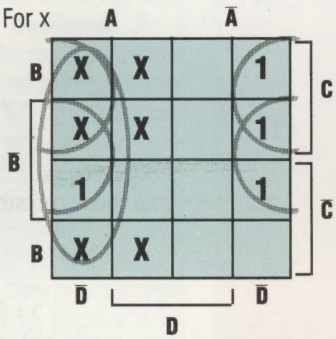
**Bar Codes**
From these diagrams we can determine which bars need to be activated to form each digit. The simple digit, '1', needs only the bars labelled 'u' and 'v' to be switched on. The truth table on the opposite page gives the outputs needed for all the digits

For w



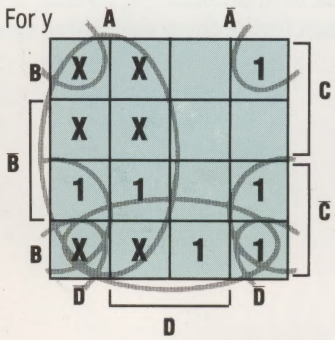

w=A+C.D̄+B̄.C+B̄.D̄+B.C̄.D.

For x



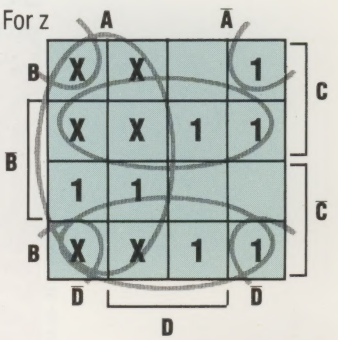

x=C.D̄+A.D̄+B̄.D̄
=(A+B̄+C). D̄

For y

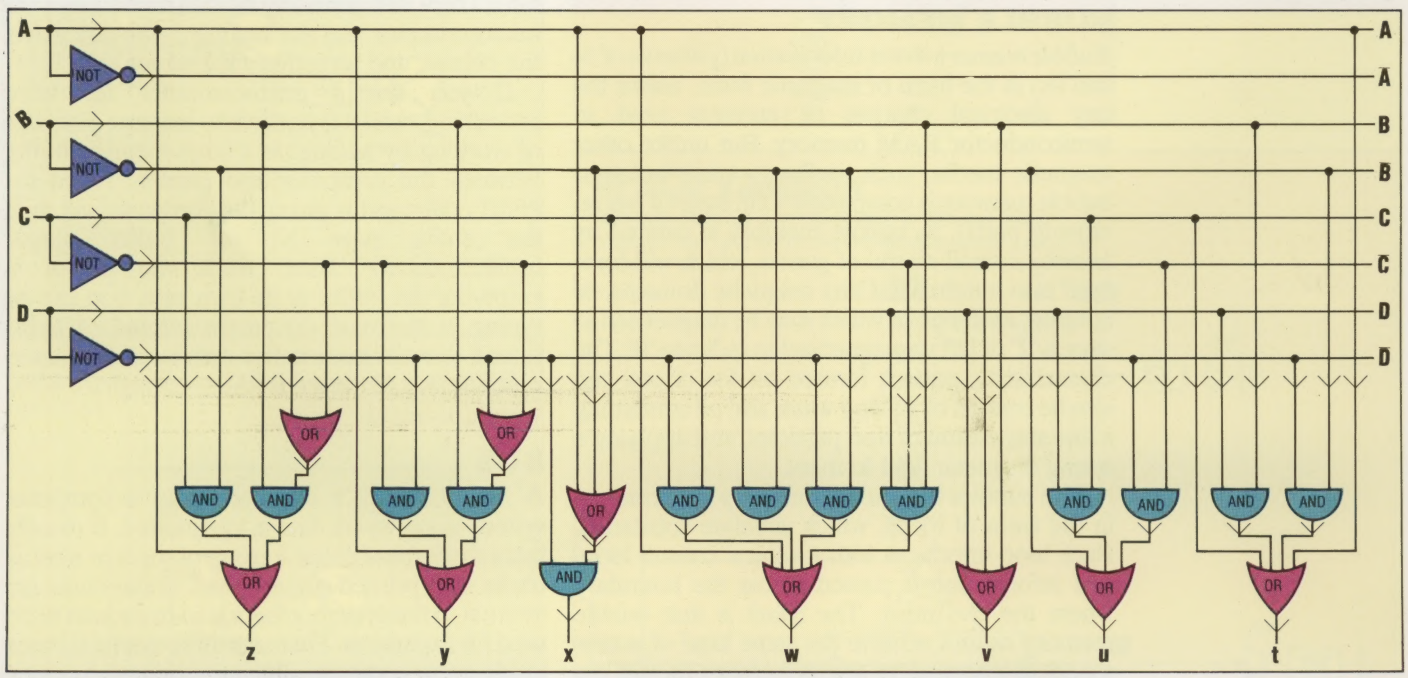



y=A+B.C̄+B̄.D̄+C̄.D
=A+B.(C+D)+C̄.D

For z




z=A+B̄.C+B.C̄+B̄.D̄
=A+B̄.C+B.(C̄+D̄)




NIXON 21

The circuit we have designed operates one display cell only. As most circuits of this type have eight or even 10 units, it would seem that it is necessary to duplicate the circuit for each cell. However, it is possible to share one converter between all the displays by a process known as *multiplexing*. This switches each unit in the display on and off in sequence so that at any one time there is only one unit accepting information from the converter circuit. Because this happens at high speed, the display appears steady and gives no trace of the rapid switching that is occurring.

# B

## BREAK

You can usually *break* a program when it is running by pressing a particular key — often STOP, BREAK or ESCAPE. A break stops a program running and returns control of the computer to the user. However, the computer will first save any temporary information used while running the program to a reserved area of memory. This will include information such as which line was being executed, any loop or subroutine counters, and so on. This is done so that you can restart the program at the point at which it was stopped without affecting its operation.

Breaks are most often used when debugging a program. You can break it at the relevant point, have a look at and perhaps alter the contents of variables and then continue program execution where you left off. Once a program is bug-free, the break key is usually disabled by the program so that it can't be stopped during execution. This reduces the risk of accidental loss of data, and protects the program listing from inquisitive eyes.

COURTESY OF NEW SCIENTIST

## BUBBLE MEMORY

*Bubble memory* stores information (patterns of 1s and 0s) in the form of magnetic force, unlike the tiny electrical charges or currents used in semiconductor RAM memory. But unlike other magnetic media, such as floppy disks or tapes, bubble memory is completely solid state (it has no moving parts). A bubble memory is created by treating a small crystal of garnet, which will form itself into hundreds of tiny magnetic domains, or bubbles, each one of which may be magnetised to store a '1' or left unmagnetised to indicate '0'. The characteristic pattern formed on the crystal can also be created by mixing water and oil containing a fine suspension of iron particles, and applying a strong magnetic field to them.

The bubbles in a bubble memory are arranged in the form of loops, which circulate constantly. Each loop interfaces with a central control loop, and information is passed across the boundary where the two meet. The result is that bubble memory cannot achieve the same kind of access speeds as true random access memory, though it is considerably faster than disk or tape. However, bubble memory does have advantages over RAM: it is non-volatile (its contents are retained even when the power is switched off) and considerably less susceptible to corruption from electrical interference.

## BUBBLE SORT

The *bubble sort* is an algorithm for sorting an array of data into alphabetical or numerical order. It is one of the easiest algorithms for the beginner to comprehend (and therefore program), but the corollary is that it is slow and inefficient compared with other techniques. The name derives from the way that, with each sorting 'pass' through the data, an item will 'bubble' towards its correct position. The same algorithm is also sometimes referred to as a *ripple* sort.

## BUFFER

The function of a buffer at the end of a railway line is to absorb the difference in speed between a moving train and a stationary end-stop, if that should ever become necessary. Buffers in computing are named by analogy with these familiar devices. There are two kinds of buffers in a typical home computer system: hardware buffers and software buffers.

A *hardware buffer* 'absorbs' the difference between the electrical operating voltages of different parts of a computer system. Most peripherals, for example, would draw more current than the output from a microprocessor or peripheral interface chip could deliver. A separate buffer chip is therefore installed, effectively just to amplify the strength of the digital signal.

A *software buffer* 'absorbs' the difference in operating speed of parts of the system. It consists of a section of RAM memory that can be filled with data by one device at its own desired speed, and then emptied by another at a faster or slower rate. There will generally be buffers between the microprocessor and the keyboard, the disk drive, the printer, and sometimes the screen.

If you use a microcomputer for word processing, then it is possible to increase your rate of working by adding on a large memory buffer between the computer and printer. When the PRINT command is given, the computer will send the entire page to the buffer almost instantaneously. Then, while the printer is emptying the buffer at its own rate, you can be typing in the next document instead of being forced to wait. Such buffer boxes can contain as much as four Kbytes of RAM — enough for a 700 word document.

## BUS

A *bus* is simply a channel within a computer system along which data is transmitted. It usually takes the form of either a group of wires or a set of tracks on a printed circuit board. Some buses are internal to the system, others lead to sockets to be used for expansion. There are three essential buses inside every computer, linking its microprocessor with the rest of the system. The *address bus* carries the address of the memory location or device to be used and the *data bus* carries information to or from that location. The *control bus* carries the signals that control and synchronise the operations of the various chips in the system.

# WORKOUT

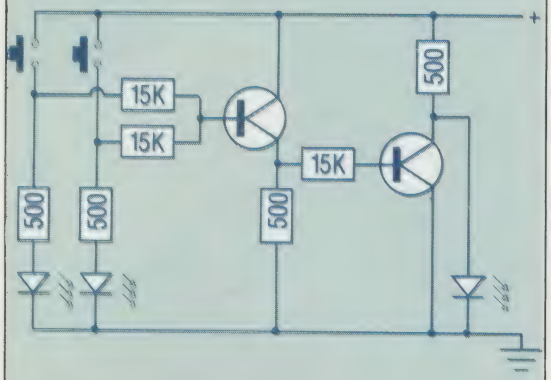
**In the last instalment of Workshop we set some simple exercises to test your knowledge of the topics we have covered. Here we present the answers. Our answers may not match yours exactly, as there are numerous ways of designing a circuit to perform a particular task. However, you will be able to prove that your circuits work by testing them against the truth tables given in the problem.**

## 1) Resistors

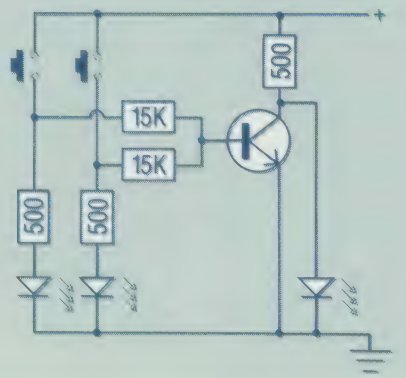
The values of the resistors illustrated are: a) 6,400K-ohm and b) 150K-ohm. A 150-ohm resistor has brown-green-brown colour bands (reading towards a gold or silver band).

## 2) NOR Gate

The most obvious method for building a NOR gate is to combine the two circuits for OR and NOT together as shown below:

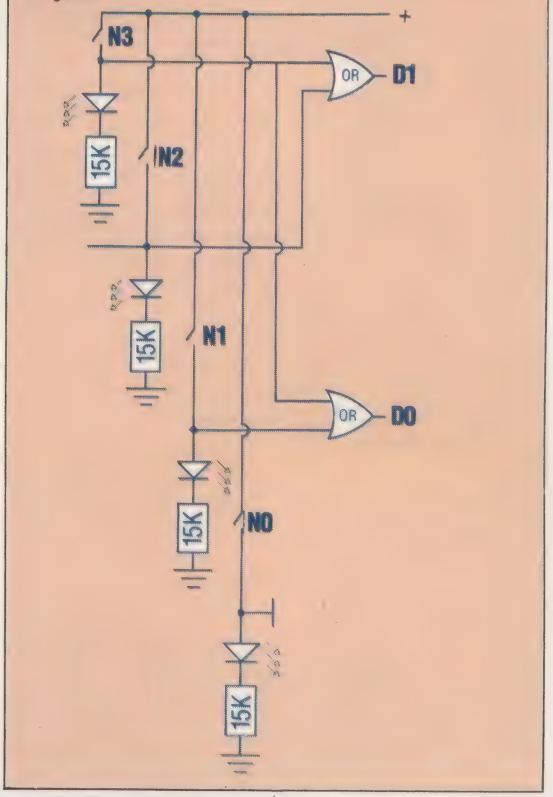



However, the short-cut method is to use the circuit for an OR gate and take the output signal from the collector of the transistor rather than its emitter, in a similar way to the NOT gate circuit:



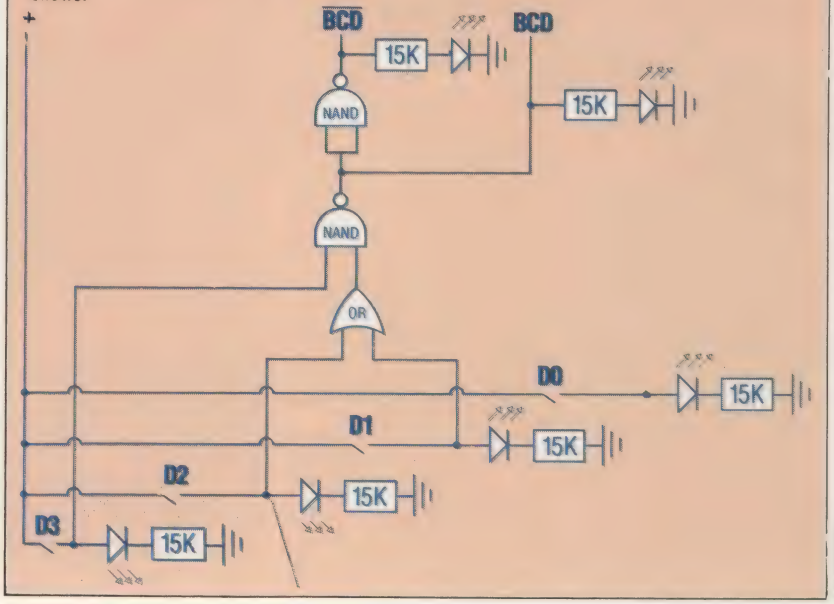

## 3) Decimal To Binary Converter

This requires only one integrated circuit — a TTL chip with four OR gates.




## 4) BCD Validator

Valid BCD codes are in the range 0000 to 1001 and invalid codes are from 1010 to 1111. We derived the necessary circuit as follows:




NOXIOUS LIZ

# MAKING MOVES

**The portable MS-DOS machine from Sharp has the appearance of a small electric typewriter. But below the lid of the PC-5000 lurk, among other features, a built-in crystal display and a bubble memory. The computer also boasts a case specially designed to accommodate its own optional printer.**

Portable computers take many forms, but the most popular style recently has been the lap-held portable, such as the TRS-80 Model 100 and the Epson HX-20. These machines incorporate a lightweight LCD screen and can run quite efficiently on battery power alone. The Sharp PC-5000 is the most expensive machine of this kind. It continues this design trend and expands into new territory by relying on bubble memory instead of cassette storage.

Bubble memory caused quite a stir in the industry when the idea was introduced several years ago, but has not caught on as expected because few companies have been able to overcome problems with speed and reliability. Sharp appears to have solved the problems, because their test machine proved to be very fast and reliable. Bubble memory requires very little power and permits the storage of large amounts of data in a tiny space. The principle involves storing data in magnetically-coded bubbles.

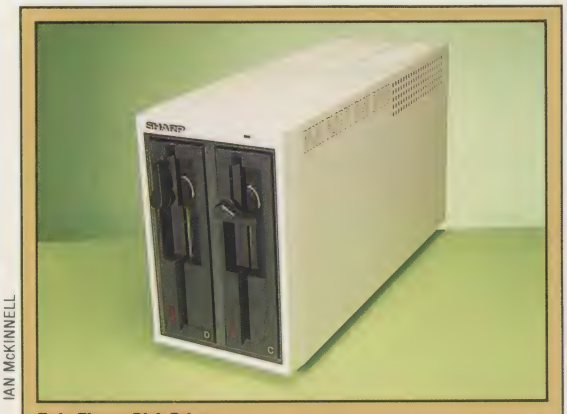
The PC-5000 resembles a small, portable electric typewriter. The lift-up lid exposes a sculptured typewriter keyboard with eight user-defined function keys and four cursor arrows placed neatly across the top. The lid is hinged and contains the LCD screen. Although fairly heavy, the lid is held in place by a ratchet assembly, which allows it to be moved into several positions for comfort. Above the keyboard is a panel with three LED warning lights, (power, low battery, bubble), and a slot for a bubble memory pack.

Behind this panel is a tray where the optional printer sits. The printer itself is a rectangular box that fits neatly into the tray. It is a thermal printer, producing 37 characters per second. On heat-sensitive paper, the print is of very good appearance, although the paper itself detracts somewhat from the quality of the document.

The Sharp PC-5000 has 128 Kbytes of user memory, and 192 Kbytes of ROM, including Microsoft GW BASIC, the same version used on the IBM PC. The CPU is Intel's 8088, again the same as the PC, and the Sharp runs MS-DOS as its operating system. The computer addresses the bubble as though it were disk drives A and B. Sharp offers a twin floppy disk drive unit that can be plugged into the back of the PC-5000. These drives would be addressed as C and D. The floppy drives cannot be run on battery power.

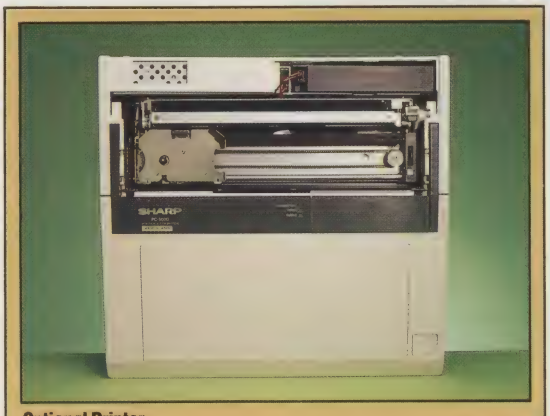
The Sharp comes with several software packages from Sorcim, including SuperCalc, SuperWriter, and SuperComm, a telecommunications program. The programs come on a bubble pack, and are chosen from the system menu by pressing one of the function keys. The values of the function keys can appear as labels on the last line of the screen.

Sharp has for years been known for quality in design and engineering, and with the PC-5000 has managed to fit a powerful computer into a small package.




IAN McKINNELL

**Twin Floppy Disk Drives**
This unit contains two 320K disk drives running under MS-DOS. It connects to the back of the PC-5000, but cannot run on the machine's battery power. Software written for other MS-DOS machines must be formatted for the 8-line display




**Optional Printer**
Looking down into the PC-5000 you can see the optional thermal printer nested in the case. The printer fits neatly into a small tray and connects to the system board via a ribbon cable. The printer is fast at 37 characters per second and produces a high quality output on heat-sensitive paper

**Bubble Memory Controller**
This card acts like a disk drive controller, except that it controls input and output from the bubble pack slot

**ROM Board**
This board has 128K ROM including MS-DOS, character generators for the display and printer, some system I/O and communications, and PISCS. PISCS is a chip that translates signals from the bubble pack into a language that MS-DOS understands. With the help of this chip, MS-DOS reads the bubble packs as disk drives A and B

**Expansion Slots**
These slots hold expansion RAM packs, or software held in ROM cartridges

**Hybrid ICs**
These ceramic integrated circuits hold the drivers for the RS232 port, disk drive connector, and power supply, among others

**8088 CPU**

**CMOS Single-chip Microcomputer**
This chip controls the operation of the entire system, and shuts down the 8088, I/O chips, etc. when they are not being used, to oonserve power

## SHARP PC-5000

**PRICE**
£1,795

**DIMENSIONS**
300 × 318 × 90 mm

**WEIGHT**
5.7 kg including printer

**CPU**
Intel 16-bit 8088 and 8-bit CMOS

**MEMORY**
128K RAM
192K ROM

**SCREEN**
80 character × 8 line LCD
640 × 80 graphics resolution.
Built-in graphics and international characters

**INTERFACES**
Cassette, external disk drive, modem output, RS232, AC power adaptor, ROM/RAM ports

**LANGUAGES AVAILABLE**
Microsoft GW BASIC

**KEYBOARD**
Standard 57-key typewriter-style with 8 function keys, cursor control keys, three other special keys. The function keys' values are software- defined, and can appear on the last line of the screen

**DOCUMENTATION**
The manuals provided include information about setting up the PC-5000 and using MS-DOS, and are quite good. The BASIC manual has been written by Microsoft and is too technical for the first-time user

**STRENGTHS**
The 5000 offers most of the facilities of a full-sized machine. The built-in printer and bubble cartridges in particular place it ahead of its rivals

**WEAKNESSES**
There are few weaknesses when you consider the developments that Sharp has made. However, the 5000 is heavy for its size and more expensive than other lap-held machines. If you are used to floppy disks, you will find the bubble memories rather slow

# FINAL REPORT

On The Market
Package Cash Book and VAT Register System
Manufacturer Dragon Data
Runs On Dragon 64
Features Maintains records of VAT transactions, as well as details of income and expenditure
Price £15
Cash Book & VAT Register System
Package Home Finance
Manufacturer BBC Soft
Runs On BBC
Price £
Package Home Financial Management
Manufacturer
Runs On
Features
Price £
HOME FINANCIAL MANAGEMENT
Package Sales Ledger
Manufacturer Kemp
Runs On Spectrum
Features Allows up to 1250 entries per month of accounts due from customers
Price £15
kemp
SALES LEDGER
Package Financial Modelling
Manufacturer Supersoft
Runs On Electron
Features Spreadsheet style package, allowing incoming and outgoing expenditure to be assessed. Data can be displayed either numerically or as bar charts
Price £44.95
FACTFINDER

IAN McKINNELL

**Analysing data is something that computers are rather good at. Even relatively simple stock recording systems, such as Dragon Data's disk-based program for the Dragon 64, are able to generate some surprisingly detailed reports. In the last article in this series on business software we look at alternative methods of analysing data.**

Anyone responsible for stock control needs to know about more than one particular item of stock. They will want to be able to look at all aspects of their stock holdings and stock movements. This can be done in two ways: either through enquiries on the screen or printed reports. The Dragon program's enquiry menu contains a number of options governing the kind of data that is available and how it will be presented.

An esssential enquiry or reporting facility concerns slow moving items. Since all transactions are dated when they are keyed in to the system, the program already has the information it needs in order to generate a slow moving items report. All it has to do is to search through the transaction history file and compare dates. Because businesses differ in their definitions of 'slow' (what is slow to one might be excellent business to another), the report has to allow the user to define the terms.

This is neatly accomplished in the Dragon system. By filling in a date in the area provided (e.g. 150484, for 15 April 1984), the user will automatically give the system a marker to start the search. All items that have no sales transaction histories will be read by the program and then printed. This provides a very useful reporting facility, since any number of slow movement reports can be generated simply by giving the program a different date. The only restriction on the user is that the date has to fall within the confines of the transaction histories on file. If there are no slow moving items that fall within the date specified, the screen message on the Dragon program will read: 'NO ITEMS WITHIN SELECTION CRITERION'.

There is one limitation to this kind of report. It only picks up stock items that have had absolutely no transactions at all. Yet it is clear that in some cases a user might well feel that a particular stock line from which there had been two sales in six months, should qualify as a slow mover.

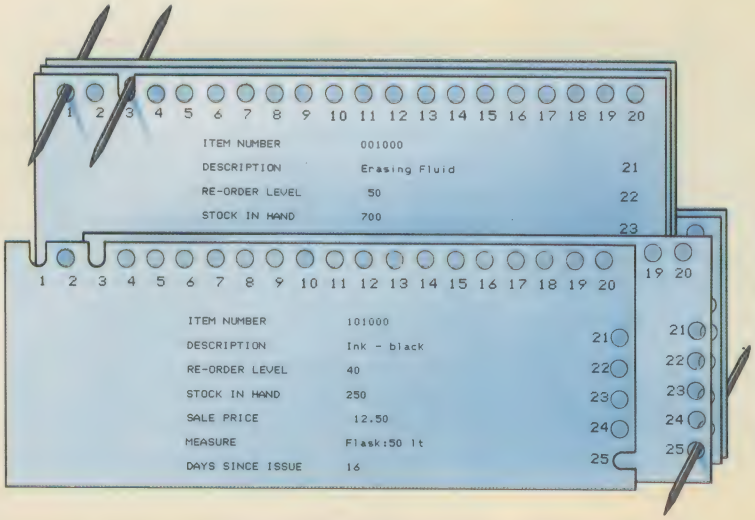
A more sophisticated system would provide greater flexibility. There are two ways of achieving this. Either the system could offer an additional selection criterion other than the date, such as specifying the number of transactions below which an item would be printed on the slow moving report. Or the system could itself read and compare movements on all the lines, listing, say, the slowest 50 lines, then the next slowest 50 lines and so on.

## SCREEN DISPLAY

The screen displays each item separately. Consequently, if there are a large number of slow moving stock items, paging through the list will be a time consuming business. In that case, therefore, a printed report might well be a better alternative than a screen display, since it is quicker and easier to scan. The systems designer has to take this into consideration when a business program such as a stock control package is being put together or 'specified'.

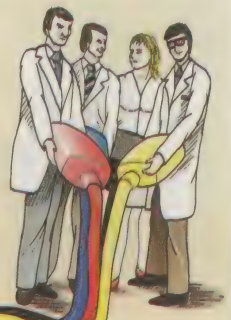
The stock movement display contains a good

## Pulling The Punches


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
ITEM NUMBER 001000
DESCRIPTION Erasing Fluid
RE-ORDER LEVEL 50
STOCK IN HAND 700
21
22
23
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
19 20
ITEM NUMBER 101000
DESCRIPTION Ink - black
RE-ORDER LEVEL 40
STOCK IN HAND 250
SALE PRICE 12.50
MEASURE Flask:50 lt
DAYS SINCE ISSUE 16
21
22
23
24
25

**Over The Edge**
The edge-punched card system is a simple form of database. The holes around the card are treated as binary digits — a hole means zero, a slot means one. Here, the cards represent a firm's stock file: holes 1 to 3 represent product group, 4 to 6 are catalogue number, so holes 1 to 6 together form the item number; holes 21 to 25 indicate the number of days since the item was last issued from stock, and hole 25 means 16 days

**Slow Movers**
Dragon Data's Stock Control System includes a database module that generates a variety of reports on the state of the inventory. Keeping large stocks of slow moving items costs money, so the firm is searching product group 101 for items not issued for 16 days or more; the search shows that stocks of printer's ink are very high, and sales are very slow, so stocks of ink must be reduced quickly



KEVIN JONES

deal of additional information that is important to the user. It shows the value of the stock, which tells the user how much capital is tied up in a warehouse or storeroom. It shows the average usage per month, the quantity in stock and the date of the last issue. From this information, the user will be able to work out a policy for shifting the stock, perhaps offering it at a large discount.

The principle of reporting according to selection criteria (the date, for example) is vital when it comes to analysing and reporting on the transaction data. Obviously, the system needs to be able to list out all transactions on all items. But it should also indicate what has happened to sections of the stock between specified dates.

The Dragon program accomplishes this by prompting the user to enter the upper and lower values of a range of stock items and the delimiting dates (e.g. between the dates 010184 and 010484).

Besides viewing all of the business's transactions on a specific range of items, the user may also want a breakdown, with items grouped together according to transaction type, or sales figures, or stock adjustments, and so on. The Dragon program has a menu with three options setting the date for the selection (these are: current period transactions only; all transactions; and transactions within a specified date range), and then a further sub-menu to allow the selection to be broken down into transaction types. This sub-menu is similar to the transaction type menu described on page 192, but it also includes a new option, '99 ALL TYPES', which allows a blanket print-out of all transactions within the data parameters specified.

The program allows users to allocate stock items to particular product groups. So users will want to review the product groups and their descriptions. They will also want to know what stocks are held for particular product groups. Screen-based reports are useful, but they are not permanent records. In order to provide a 'hard-copy' record, the reporting facility duplicates in many respects the enquiry facilities offered by this program.

There are a number of reasons why the Dragon program is a rather restricted stock control system, despite its detailed analytical features. The system is designed as a 'stand-alone' stock system. It is not able to link up with other, related business applications programs that could use the information on this program's files. There is no facility for allocating stock against orders. One of the most important questions users want answered is: 'Have I sufficient stock in hand to meet these orders?' Once you have several orders from customers with a number of different stock items requested on each order, keeping track of the demands made on stock is very difficult manually.

Throughout this series, we have concentrated on the use of home computers for small business data processing. It should by now be clear that, although these packages may offer supposedly integrated systems, this often means that they concentrate on one aspect of the business — such as cash flow — at the expense of the others — order processing, inventory, reconciliation, for example. Such packages are perhaps best used to provide supplementary information to business managers.

Despite these shortcomings, home computer business packages can greatly improve the efficiency of small business accounting and organisation. If chosen and used with care, such packages can prove very useful indeed.

# GO SUB GO!

**This short series of articles is designed to help you get the most from the graphics capabilities of the Commodore 64. We will cover many aspects, including screen displays, designing sprites, and keyboard control of movement. During the course, we will construct a Subhunter game, building up the necessary routines as we go along.**

One of the Commodore 64's most attractive features is its ability to allow arcade-type games to be written in BASIC. What makes this possible is the machine's ability to handle sprites — high resolution shapes that can be defined by the user and easily manipulated on the screen. There are special registers in the Commodore's memory that control attributes of the sprites, such as their colour, size and position on the screen. By POKEing numbers into these registers, the programmer can easily control the action. The Subhunter game that we will design as the focal project of this series will make use of four of the machine's eight available sprites. These sprites will represent the ship, the submarine, the depth charges fired from the ship, and an explosion.

So that the game can be built up over successive instalments, each section of the program will be written as a short subroutine that is called up for use within the main program loop. This type of structure makes tracing bugs and program extension much easier.

The concept of this game will be familiar to many. The player is in control of a ship that is hunting submarines. These cross the screen at varying depths and speeds, and the ship drops depth charges on them. The player's score is increased every time a sub is hit, the value of each sub being calculated from its depth and speed. Naturally, higher scores are made for hitting deep, fast-moving submarines. However, if the sub escapes, then its value is subtracted from the player's score. The ship is controlled from the keyboard, using the Z and X keys for horizontal movement left and right. Depth charges are fired using the M key. A timer is displayed on the screen, allowing the game to be played for three minutes. At the end of this time the player is asked if another game is required, and a record is kept of the highest score since the program was first run.

The golden rule to observe when designing an action game in BASIC is to keep the main loop of the program as short as possible. The Subhunter program makes use of subroutines to carry out most functions within the game. The only functions controlled directly from within the main program loop are: updating the timer, accepting an input from the keyboard, moving the ship, and moving the submarine.

This program design is general enough to be applied to any make of computer, but the detailed programming will vary according to the individual characteristics of the computer being used. In this section of the project we shall look at the routine that creates a screen display.

## SCREEN DISPLAY

There are two ways to print characters to the screen on the Commodore 64. One is to use the PRINT statement and the other is to POKE numbers to the areas of memory that hold information about the display. We will use both methods in the creation of the backdrop to our game.

The Commodore screen is made up of 25 rows of 40 character spaces. In other words, there are 1,000 places on the screen where a character can be placed. Each position on the screen has two numbers associated with it. The first is a screen code number that tells the computer which character to display in that position. The second is a colour number that tells the computer what colour the character displayed should be. There are two blocks of memory, each consisting of 1,000 locations: one to hold information about the screen code and one to hold information about the colour of every position on the screen. The area that holds the screen codes is called the screen memory and runs from location 1024 to 2023. The colour memory runs from 55296 to 56295.

Each character has its own screen code and these are listed on page 132 of the user guide. There are 16 colours on the Commodore 64. The colour codes are:

| | | | |
|---|---|---|---|
| 0 | black | 8 | orange |
| 1 | white | 9 | brown |
| 2 | red | 10 | light red |
| 3 | cyan | 11 | grey 1 |
| 4 | purple | 12 | grey 2 |
| 5 | green | 13 | light green |
| 6 | blue | 14 | light blue |
| 7 | yellow | 15 | grey 3 |

In addition to these areas of memory, two other locations are of special interest. Location 53280 controls the border colour and 53281 controls the screen colour. To design the seascape setting for our game, the top six rows of the screen will be light blue for sky, while the rest of the screen and the border will be dark blue to represent the sea (except for the bottom two rows, which will represent the sea bed).

To set the screen colour to light blue and the border colour to dark blue, the following pair of POKE commands is required.

```
POKE53281,14:POKE53280,6
```

The seventh row of the screen starts at location 1264. The second row from the bottom starts at location 1944. We can colour the sea by POKEing the screen code for a reverse space (a space character that is blocked-out) into locations 1264 to 1943, and POKEing 6 into the corresponding colour memory locations. There is an easy way to connect a colour memory location with a corresponding screen memory location: simply add 54272 to the screen memory location.

The screen code for a space character is 32. A reverse character's screen code can be calculated by adding 128 to the normal screen code, so the code for a reverse space is 32 + 128 = 160. The following few lines of program make use of a simple FOR...NEXT loop to colour the sea:

```
FOR I=1264 TO 1943
POKE I,160:POKE I+54272,6
NEXT I
```

The seabed consists of two rows of a chequered character with a screen code of 102, coloured brown. Again a simple FOR...NEXT loop will do this:

```
FOR I=1944 TO 2023
POKE I,102:POKE I+54272,9
NEXT I
```

The PRINT statement is also an effective method of producing screen displays. Colour and cursor positioning can be controlled from within a PRINT statement either by using the special Commodore control characters or by using CHR$ codes. We will use the latter method, as that is easier to read in program listings. A full list of CHR$ codes is given in Appendix F of the user guide. We are interested in those that affect colour and cursor position:

| | |
|---|---|
| CHR$(5) | colour white |
| CHR$(144) | colour black |
| CHR$(147) | clears screen and positions cursor in top left corner |
| CHR$(19) | positions cursor in top left corner |
| CHR$(17) | moves cursor one place DOWN |
| CHR$(145) | moves cursor one place UP |
| CHR$(157) | moves cursor one place LEFT |
| CHR$(29) | moves cursor one place RIGHT |

As part of our screen setup routine SCORE and HI SCORE must be PRINTed on the top line of the screen. CHR$(19) will ensure that the cursor is at the beginning of the top line. The following command PRINTs the initial score in black:

```
PRINT CHR$(19);CHR$(144);"SCORE 000"
```

The HI SCORE is also to be positioned on the top line, but on the right-hand side. The SPC function allows a number of spaces to be inserted. The PRINT command can now be altered to include the HI SCORE:

```
PRINT CHR$(19);CHR$(144);
   "SCORE 000";SPC(16);"HI SCORE 000"
```

The screen setup routine will form a subroutine to the main program, starting at line 1000. Also included is a POKE command that causes all the keys on the keyboard to repeat when they are pressed. This will be used later when the keyboard control routine is discussed. This subroutine can be tested by the following program lines:

```
10 GOSUB 1000: REM SCREEN SETUP
20 END

1000 REM **** SCREEN SETUP ****
1010 PRINT CHR$(147):REM CLEAR SCREEN
1020 :
1030 REM ** COLOUR SEA **
1040 POKE53281,14:POKE53280,6
1050 FOR I=1264TO1943
1060 POKE I,160:POKE I+54272,6
1070 NEXT
1080 :
1090 REM ** SEA BOTTOM **
1100 FOR I=1944TO2023
1110 POKE I,102:POKE I+54272,9
1120 NEXT
1130 POKE 650,128:REM REPEAT KEYS
1140 :
1150 REM ** SCORE**
1160 PRINTCHR$(19); CHR$(144);
     "SCORE 000";SPC(16);"HI SCORE 000"
1170 RETURN
1180 :
1190 :
```

Once the routine has been entered, it is a good idea to save it on tape or disk before running it.

**Subhunter In Action**
The program we are developing in this series is the classic computer graphics game of depth-charging submarines. The Commodore 64's sprite grahics are used to provide smooth animation of the sub and ship

IAN McKINNELL

# COUNTER INSTRUCTIONS

**Loops and conditional branches are implemented in Assembly language by using the processor status register flags to test the condition of the accumulator, and the relative jump instructions to change the flow of control in the program. These structures and the indexed addressing mode combine in creating data tables.**

Before we can begin to use the various CPU addressing modes (especially indexed addresses) to advantage, we must first be able to write a loop. Without this fundamental structure we are in much the same position as a BASIC programmer who knows about arrays, but is ignorant of the FOR...NEXT command. There are no automatic structures like FOR...NEXT in Assembly language (though there is a Z80 instruction that is very close to it), but we can construct loops of the IF...THEN GOTO... type. These require instructions that make decisions or express conditions, and effectively change the order in which instructions are obeyed in the program.

Decision making in Assembly language centres on the flags in the processor status register. These flags show the effects on the accumulator of the last instruction executed, and are sometimes called *condition flags*. All these flags can be used in decision making, but we will need to consider only two of them at present — the zero (Z) and the carry (C) flags.

The state of these flags can be used to decide whether the processor executes the next instruction in the program, or whether it jumps to another instruction elsewhere in the program. The decision to continue or to jump is arrived at by the processor's either changing or accepting the address contained in its program counter. This register always contains the address of the next machine code instruction to be obeyed. When the processor begins to execute an instruction, it loads the op-code of the instruction from the byte pointed to by the address in the program counter. The address in the register is incremented by the number of bytes in the instruction so that the program counter then points to the op-code of the next instruction. If the current instruction causes the program counter to point to an address elsewhere in the program, then a jump is effectively generated.

On the 6502, the instruction BEQ causes the program counter to be changed if the zero flag is set. BCS is the equivalent instruction if the carry flag is set. On the Z80, these instructions are JR Z and JR C respectively. These four op-codes are called *branch instructions* because they represent a branch-point in the flow of program control. Their operand is a single-byte number, which is added to the address in the program counter to produce a new address. Consider what happens when the following program is executed:

| ORG $5E00 | | | |
|---|---|---|---|
| 6502 | | Z80 | |
| 5E00 | ADC #$34 | ADC | A,$34 |
| 5E02 | BEQ $03 | JR | Z,$03 |
| 5E04 | STA $5E20 | LD | ($5E20),A |
| 5E07 | RTS | RET | |

If the ADC instruction at $5E00 produces a zero result in the accumulator (which is unlikely but, as we'll see later, possible), then the BEQ and JR Z instructions at $5E02 will cause $03 to be added to the contents of the program counter. The next instruction to be executed, therefore, will be the return instruction at $5E07, causing the instruction at $5E04 to be skipped over.

At first sight, this may seem wrong. After all, if the instruction at $5E02 causes $03 to be added to the program counter, surely the address stored there will become $5E05? But it is important to remember that the program counter always points to the *next* instruction to be executed and not the instruction currently being obeyed. Thus, when the instruction at $5E02 begins execution, the program counter will contain the address $5E04 — the location of the next instruction. If $03 is added to $5E04 the result will be $5E07, the address of the following instruction.

It's worth remarking here that the processor is not capable of checking whether the addresses pointed to are correct. If we inadvertently change the displacement in the instruction to $02, then the program counter will be increased (if the accumulator contains zero) by $02, and the processor will consider $5E06 to be the address of the op-code of the next instruction. In our correct program, $5E06 contains the value $5E, which is the hi-byte of the operand of the instruction at $5E04. The processor, however, cannot evaluate whether it is the right instruction or not. As far as it is concerned, $5E is a valid op-code and it will proceed to execute it, taking the bytes following $5E06 as the operands of the instruction. The program will probably crash as a result. Miscalculating displacements like this is one of the commonest errors in machine code programming.

In Assembly language programming, however, calculating jump displacements need not be a problem because the assembler program can do it for us. Therefore, instead of supplying a hex displacement as the operand of the branch

instruction, we give the symbolic address of the instruction to be jumped to. This makes the Assembly language program far easier to follow. The assembler decodes the symbolic address into an absolute address, calculates the displacement necessary to get to the address, and writes that displacement into the machine code instruction. The symbolic address is called a *label*, and it's analogous to a BASIC program line number.

Let's take a closer look at how labels are used. A label is an alphanumeric string written at the start of an Assembly language instruction. It is treated by the assembler program as a two-byte symbol standing for the address of the first byte of the instruction. Therefore, we can re-write the program given in this way:

| ORG $5E00 | | |
|---|---|---|
| | 6502 | Z80 |
| 5E00 | ADC #$34 | ADC A,$34 |
| 5E02 | BEQ EXIT | JR Z,EXIT |
| 5E04 | STA $5E20 | LD ($5E20),A |
| 5E07 EXIT | RTS | RET |

The instruction at $5E02 can now be read as 'IF the value of the accumulator is zero THEN GOTO the address represented by the label EXIT'. This is an enormous improvement in readability over the previous version, and greatly decreases the chance of miscalculating the jump destination.

We can now use labels and the branch instructions to create a loop:

| ORG $5E00 | | |
|---|---|---|
| | 6502 | Z80 |
| 5E00 START | ADC #$34 | ADC A,$34 |
| 5E02 | BNE START | JR NZ,START |
| 5E04 | STA $5E20 | LD ($5E20),A |
| 5E07 EXIT | RTS | RET |

Notice here the use of the new label, START, as well as the new branch instructions: BNE, meaning 'Branch if the accumulator is Not Equal to zero'; and JR NZ, meaning 'Jump if the accumulator is Not equal to Zero'. Let's consider what effect this code will have. The program will first add $34 to the accumulator. If the result is not equal to zero then the program branches back to $5E00 — the address represented by the label START. $34 will again be added to the accumulator, and the result will decide whether another branch occurs. This 'loop' will go on and on until the branch condition is met. When the contents of the accumulator do equal zero following an ADC instruction, then the branch at $5E02 will not occur, and the instruction at $5E04 will be executed next.

This is exactly like an IF...THEN GOTO... loop in BASIC, except that it's difficult to see how the accumulator could ever become zero. After all, it is being increased by $34 every time the loop is executed! How will it ever add up to zero? The answer lies in the fact that the accumulator is only a single-byte register, and if the addition results in a two-byte number, then the carry flag of the processor status register will be set, and the accumulator will hold the lo-byte of the result. If the accumulator contains $CC, for example, then adding $34 will give the two-byte number $0100. The carry flag will be set, and the accumulator will hold the lo-byte of this result — $00. Thus, the contents of the accumulator would be zero, and the zero flag set as a result.

With this result in mind, we might re-write the program to use a different branch condition, incorporating the state of the carry flag rather than the state of the zero flag.

| ORG $5E00 | | |
|---|---|---|
| | 6502 | Z80 |
| 5E00 START | ADC #$34 | ADC A,$34 |
| 5E02 | BCC START | JR NC,START |
| 5E04 | STA $5E20 | LD ($5E20),A |
| 5E07 EXIT | RTS | RET |

In this version, the instruction at $5E02 reads 'if the carry flag is clear, branch to START'. As soon as the result of adding $34 to the accumulator is greater than $FF, then the carry flag will be set, and the branch back to the START address will not occur.

## LOOP COUNTERS

It may seem that branching according to the current condition of either the carry flag or the zero flag is a rather limited facility, but it permits a wide range of decision making, as we shall shortly see. What is definitely lacking from our repertoire now is the ability to keep a *loop counter*. We might wish, for example, to count the number of times that a loop is performed before the exit condition occurs, or we might want to cause an exit from the loop after a given number of iterations. The first objective is easily achieved by employing a CPU index register to hold the counter, and an increment instruction to update the counter:

| 6502 | | |
|---|---|---|
| 0000 | ORG | $5DFD |
| 5DFD | LDX | #$00 |
| 5DFF START | INX | |
| 5E00 | ADC | #$34 |
| 5E02 | BCC | START |
| 5E04 | STX | $5E20 |
| 5E07 EXIT | RTS | |

| Z80 | | |
|---|---|---|
| 0000 | ORG | $5DFA |
| 5DFA | LD | IX,$0000 |
| 5DFE START | INC | IX |
| 5E00 | ADC | A,$34 |
| 5E02 | JR | NC,START |
| 5E04 | LD | ($5E20),IX |
| 5E08 EXIT | RET | |

The new structure has forced several changes in the program. Firstly, the instructions inserted at the beginning of the program require a new ORG address. These instructions have much the same effects on both the 6502 and the Z80 processors, but their lengths are different, so the location addresses are no longer the same in both versions of the program.

Secondly, new versions of the load (LDX, LD IX) and store (STX, LD( ),IX) instructions have been used to place an initial value of $00 in the CPU

index register. The 6502 X register is a single-byte register, but the IX register of the Z80 has two bytes. The index registers have special functions, but they are essentially CPU RAM just like the accumulator, and here we use them as extra accumulators in which to keep the loop count. When the loop exit occurs, the contents of the 6502 X register will be stored at $5E20. In the Z80 version the lo-byte of the (two-byte) IX register will be stored at $5E20 and the hi-byte at $5E21.

Thirdly, a completely new instruction has taken the place of the ADC instruction as the START of the loop: INX and INC IX are both increment instructions, causing the contents of the index register to be increased (or incremented) by $01. This updates the value of the loop counter every time the loop is executed.

We can see the program as reading: 'make the loop counter zero, start the loop by incrementing the counter, add $34 to the accumulator, and branch back to the start of the loop if the carry flag is clear, otherwise store the loop counter contents at $5E20'. A further modification of the program will greatly increase its usefulness and scope:

**6502**

| | | |
|---|---|---|
| 0000 | ORG | $5DFA |
| 5DFA | LDX | #$00 |
| 5DFC START | STA | $5E22,X |
| 5DFF | INX | |
| 5E00 | ADC | #$34 |
| 5E02 | BCC | START |
| 5E04 | STX | $5E20 |
| 5E07 EXIT | RTS | |

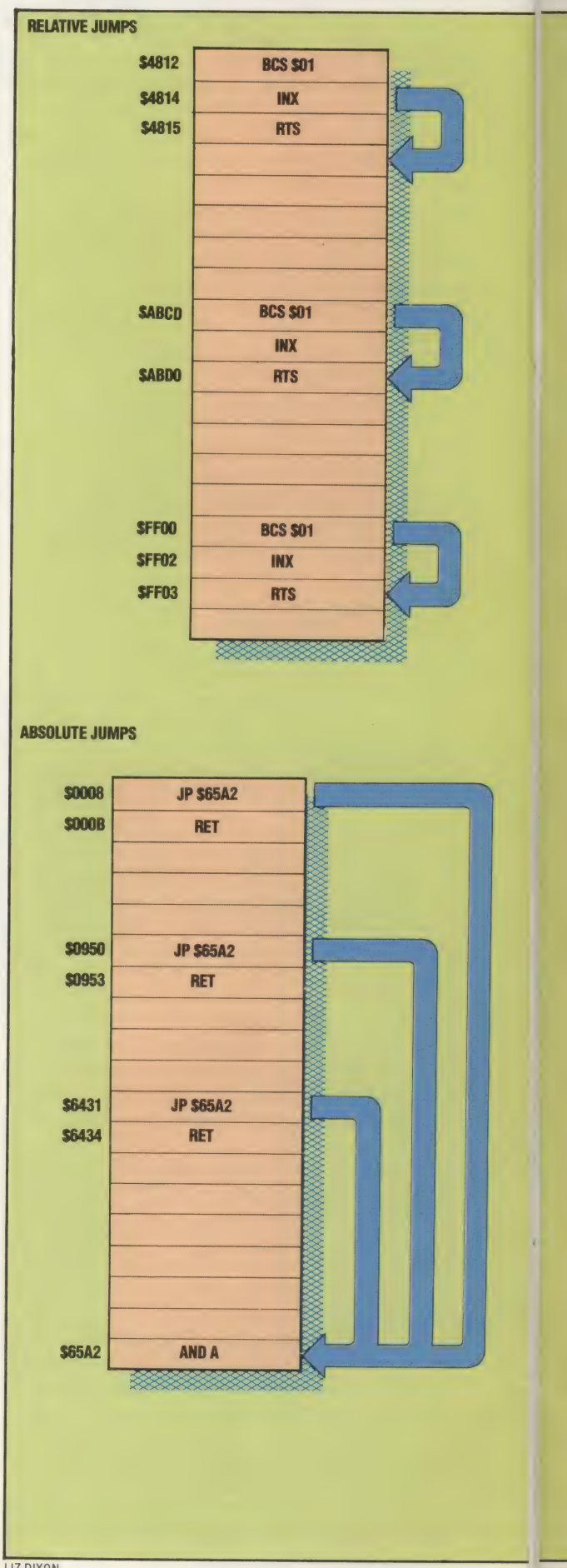
**Z80**

| | | |
|---|---|---|
| 0000 | ORG | $5DF7 |
| 5DF7 | LD | IX,$5E00 |
| 5DFB START | LD | (IX+$22),A |
| 5DFE | INC | IX |
| 5E00 | ADC | A,$34 |
| 5E02 | JR | NC,START |
| 5E04 | LD | ($5E20),IX |
| 5E08 EXIT | RET | |

The 6502 and Z80 versions both have the same effect: they create at location $5E22 a storage table of the successive values of the accumulator as the program is executed, and eventually store at $5E20 the final value of the loop counter, which is also the number of bytes in the table starting at $5E22.

The 6502 version achieves this through the instruction STA $5E22,X, which means 'add the contents of the X register to the base address, $5E22, then store the contents of the accumulator at the address thus formed'. The STA instruction is here in the absolute direct indexed mode: that is to say, the X register is used as an index to modify the base address, $5E22. Since the X register is initialised to $00 and subsequently incremented every iteration, the starting value of the accumulator will be stored at $5E22, the next value at $5E23, and so on. After the loop exit occurs, STX will store the final value of the loop counter at location $5E20.

The Z80 version uses the IX register as a pointer to the current storage address, while still using the




LIZ DIXON

**Relative Jumps**
Most of the branch instructions, such as BCS (meaning 'branch if carry flag is set'), JR NZ (meaning 'branch if the accumulator is non-zero'), act according to the condition of the processor status register, and use the relative jump mode in redirecting the flow of control through the program. The alternative is the absolute jump.

In the example, the BCS $01 instruction always causes a relative jump of one byte forward (when it causes a jump at all, that is; it's conditional on the state of the carry flag) no matter what the location address at which the machine code resides. Here, the BCS $01 instruction is always followed by the INX instruction, itself only a single-byte instruction; when the carry flag is set, therefore, BCS will cause the INX instruction to be skipped.

**Absolute Jumps**
In this example, the JP $65A2 instruction causes an unconditional jump whenever it is encountered. Its effect is to redirect program execution to the address which forms its operand — $65A2 here. No testing is done, and the location address of the instruction at the time of execution is not significant; program execution always continues from the specified address.

Both jump modes have advantages and disadvantages, but the most important criterion in choosing between a relative jump or an absolute jump is relocatability: it's quite common in Assembly language programming to write a routine and assemble it at one ORG address, then re-use it in the same form but with a different ORG value. If all the jumps in the routine are relative, then changing the location addresses of the instructions will not matter at all, and the program will flow smoothly along its intended paths; if any of the jumps is absolute, however, when the routine is assembled at a different ORG, the jumps will still send control to the specified address, which may now have no significance for the routine. Relative jumps are relocatable, absolute jumps are not.

lo-byte of IX as the loop counter. The instruction LD IX,$5E00 puts the base address, $5E00, into the IX register, so the lo-byte of IX will contain $00. The peculiar-looking instruction LD (IX+$22),A means 'add the address contained in IX to $22, and store the contents of the accumulator at the address thus formed'. Since IX is initalised to $5E00, and is subsequently incremented at every loop interation, the starting value of the accumulator will be stored at $5E22, the next value at $5E23, and so on. Meanwhile the lo-byte of IX records the number of loop iterations, and is finally stored at $5E20 when the loop terminates. The LD (IX+$22),A instruction here is in the absolute indirect indexed addressing mode, which is rather more complicated than the 6502 version but much more powerful.

We have now looked at the Assembly language loop and array structures in some detail. These are both extremely helpful machine code programming techniques. In the next instalment of the course, we'll put them both to work.

## Exercises

There are many important, and possibly puzzling, points in this instalment, and only experience of using the new addressing modes and instructions will make you fully understand them.

Use the CHAMP assembler package to assemble and SAVE the various program fragments in this instalment. When you execute a fragment, use the ‹debug› mode to examine the memory locations that should be affected. It's a good idea always to initialise these locations with a recognisable constant — $FF, for instance — before execution, so that afterwards you can tell whether memory has been affected by the program. You can use the ‹debug› Alter command to do this, or even the ‹debug› Move command.

**Remember, as always, that the location addresses given in the program are for example only, and that you must choose addresses suitable for your machine.**

**Loading And Saving CHAMP**
For convenience and security you should copy CHAMP onto another tape, and then remove the write-protect tabs from the original and the copy. In the following instructions, the LOAD instructions refer to the CHAMP tape, and SAVE refers to the copy tape:

**BBC Model B**
1) LOAD"CHAMP"
2) SAVE"CHAMP" : RUN : Quit to BASIC
3) *SAVE"CHAMP M/C" 1000 , 4600

**Commodore 64**
1) LOAD"CHAMP"
2) SAVE"CHAMP" : RUN : enter <debug> mode
3) Hit [w][ret], followed by [s] for SAVE
4) Start address 1000; end address 4600; filename "CHAMP M/C"

**Spectrum**
1) LOAD"CHAMP"
2) Quit to BASIC : SAVE "CHAMP" LINE 1
3) SAVE "CHAMP M/C" CODE 27000,9231

The conditional branch instructions, as we have seen, depend on the contents of the processor status register. One reason for adding the binary display option to the Monitor program (see pages 118 and 198) was to enable you to inspect the contents of the PSR before and after an instruction is executed, and observe the changes in the flags. There is no single instruction in either 6502 or Z80 Assembly language to store the PSR contents, so we must use these commands:

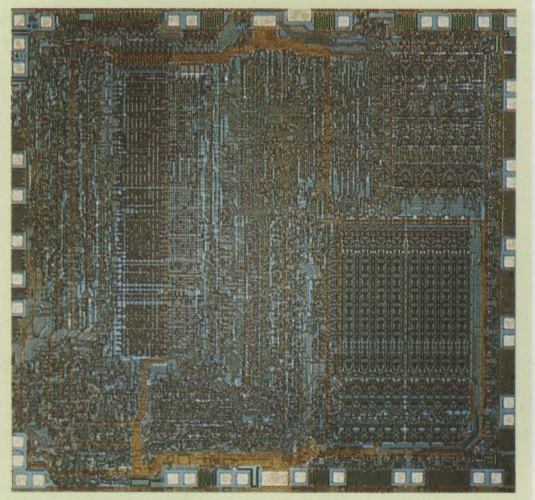
| Z80 | |
|---|---|
| 3E00 F5 | PUSH AF |
| 3E01 F5 | PUSH AF |
| 3E02 E1 | POP HL |
| 3E03 22 lo hi | LD (STORE1),HL |
| 3E06 F1 | POP AF |
| **6502** | |
| 3E00 48 | PHA |
| 3E01 08 | PHP |
| 3E02 48 | PHA |
| 3E03 08 | PHP |
| 3E04 68 | PLA |
| 3E05 8D lo hi | STA STORE1 |
| 3E08 68 | PLA |
| 3E09 8D lo' hi' | STA (1+STORE1) |
| 3E0C 28 | PLP |
| 3E0D 68 | PLA |

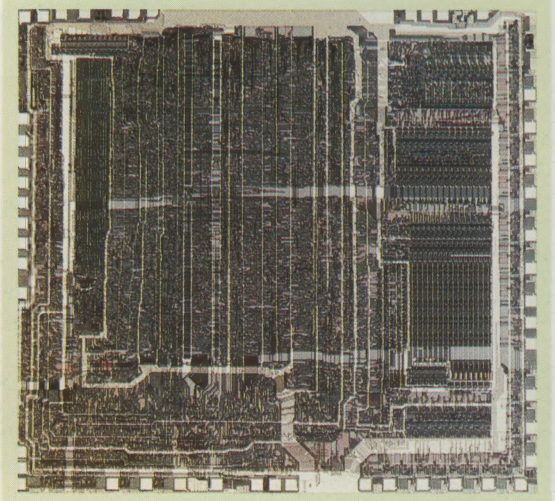
This sequence of instructions will cause the current contents of the PSR to be stored in the byte addressed by STORE1 (an address appropriate to your machine), while the accumulator contents will be stored at (1+STORE1). To use these instructions, simply insert them as a block before and after the program instruction whose effect you wish to observe. You must remember, however, to add two to the value of STORE1 every time you insert this block. When you've executed the program, you can use the Monitor to display the section of memory where you've stored the various contents of the PSR and the accumulator.

It may occur to you that this block should be treated as a subroutine rather than repeatedly entering it where it is required. There is an Assembly language equivalent of BASIC's GOSUB, but using it here would complicate matters since it uses the stack, and this would interfere with the block's use of the same list (PLA, PUSH, PHP, etc. are all stack manipulations, which will be more fully explained later). You may notice the difference in length between the Z80 and 6502 code: the Z80's two-byte registers and associated instructions are responsible for this variation.

# BREAKING AWAY

**Chip Set**
These photo-micrographs illustrate the complexity of microprocessor circuits. On the left is a highly magnified picture of Zilog's immensely successful Z80 and on the right, a more recent chip from the Z8000 series. Notice that the Z8000, being a 16-bit chip, has a more dense design and more connections around the edge of the chip





**When Zilog Incorporated first introduced the Z80 microprocessor in 1977, few people suspected that it was about to begin a revolution. But within a few short years, the Z80, and its arch rival the 6502, would turn into reality what had previously been considered a flight of science fiction fantasy — a computer in every home.**

The history of Zilog begins in the early seventies when Frederico Faggin and Masatoshi Shima, two employees of the microchip manufacturer Intel, broke away to form their own company. The two men had been involved in the development of the 8080A microchip (considered the first 'computer on a chip'), and using this experience they began work on a new development of the microprocessor. The 8080 was already proving to be very popular with computing designers and hobbyists, and so Faggin and Shima, not unnaturally, decided to design the new chip to be compatible with the 8080. It could therefore take advantage of the large quantity of software that had already been written for the 8080. Using their detailed knowledge of the 8080, they were able to extend the instruction set (the list of machine code commands contained within the microprocessor) by introducing extra registers, two - byte op-codes and other techniques. The microprocessor — the Z80 — was a considerable improvement.

This revolution in hardware fortunately coincided with a parallel revolution in software. In 1972, Gary Kildall and John Torode wrote a program called Control Program/Monitor, or CP/M, which allowed a microprocessor to handle the recently introduced floppy disk. Because Kildall was a consultant to Intel, the program was designed to run on the 8080 and 8085. CP/M was rapidly becoming the dominant disk handling system for microcomputers, and with their powerful new 8080-compatible microprocessor, Zilog were ideally placed to take advantage of the rush to CP/M software.

Zilog's path has not been so smooth in subsequent years. Although the Z80 is still selling in vast quantities — the company still produces around a million a month — attempts to upgrade the chip to the 16-bit market have met with a mixed response.

Zilog's first attempt at a 16-bit processor was the Z8000. Although generally acknowledged as a very powerful device, with a comprehensive instruction set and a large number of registers, it proved to be an extremely complex chip to program. The Z8000 encountered several other major obstacles. To begin with, although compatible with the yet to be launched 32-bit Z80000 (or Z80K), it was not compatible with the Z80 and so was unable to take advantage of the range and variety of programs that had been written for the Z80 in the preceding years. This meant that those manufacturers interested in 16-bit upgrades for their machines tended to turn to a less demanding microchip.

With the Z8000 proving to be unpopular in the microcomputer market, Zilog went back to the drawing board. The company is soon to launch the Z800 16-bit processor, which is compatible with the Z80. Things are also looking bright in other areas: Commodore have announced that their new range of business machines will be powered by the Z8000.

**Zilog's President, Frank de Weeger**

# DATA IN STORE

## Coming soon in THE HOME COMPUTER ADVANCED COURSE

### Making Waves

Beautiful patterns can be created with ease by using the right geometrical formulae. We show you how with a program for the BBC Micro, Spectrum and Oric-1/Atmos in **Issue 17**

### Travelling Light

Computer buffs can take their enthusiasm travelling and impress their fellow train passengers if they equip themselves with a lap-held portable micro. Our feature in **Issue 18** reviews half a dozen of these light-weight wonders

### Flower Power

The choice of the right printer can be one of the most confusing topics that the computer user has to face. In **Issue 19** we shed light on daisy wheel printers, which offer the highest quality of print – but have snags, too

### Tread Lightly

Minefield is a favourite computer game calling for quick wits and a fast finger on the keyboard. In a new series beginning in **Issue 20**, we program this game for the BBC Micro, taking advantage of the machine's superb graphics capability

### Power To The People

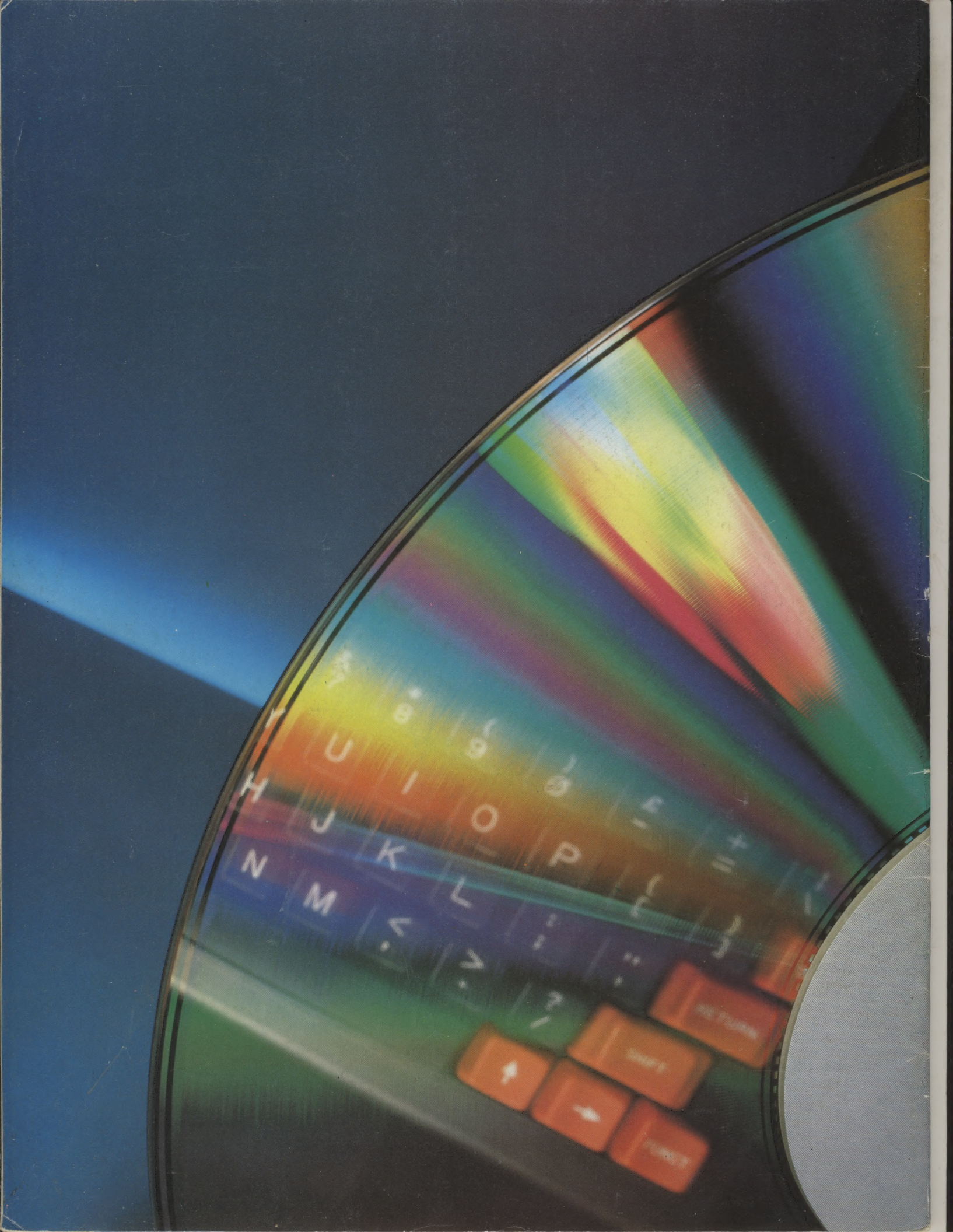
If computing leaves you with no money to invest on the stock exchange, do the next best thing by playing a business game on your micro. We review a portfolio of these simulations in **Issue 21**

### Trace Elements

Give your computer some pictures to look at by linking it to a digitiser – a device that turns images into numbers. Digitisers are explained and compared in **Issue 21**

All this and, of course, the *Home Computer Advanced Course* regular features: applications, reviews of hardware and software, programming techniques, machine code programs, and much more...

PLACE A REGULAR ORDER WITH YOUR NEWSAGENT


U I O P
H J K L
N M